NAME

bt_gethostbyname, **bt_gethostbyaddr**, **bt_gethostent**, **bt_sethostent**, **bt_endhostent**, **bt_getprotobyname**, **bt_getprotobynumber**, **bt_getprotoent**, **bt_setprotoent**, **bt_endprotoent**, **bt_aton**, **bt_ntoa**, **bt_devaddr**, **bt_devname**, **bt_devinfo**, **bt_devenum**, **bt_devopen**, **bt_devclose**, **bt_devsend**, **bt_devrecv**, **bt_devreq**, **bt_devfilter**, **bt_devfilter_pkt_set**, **bt_devfilter_pkt_clr**, **bt_devfilter_pkt_tst**, **bt_devfilter_evt_set**, **bt_devfilter_evt_clr**, **bt_devfilter_evt_tst**, **bt_devinquiry**, **bt_devremote_name**, **bt_devremote_name_gen**, **bdaddr_same**, **bdaddr_any**, **bdaddr_copy** - Bluetooth routines

LIBRARY

Bluetooth Library (libbluetooth, -lbluetooth)

SYNOPSIS

#include <bluetooth.h>

*struct hostent **
bt_gethostbyname(*const char *name*);

*struct hostent **
bt_gethostbyaddr(*const char *addr, int len, int type*);

*struct hostent **
bt_gethostent(*void*);

void
bt_sethostent(*int stayopen*);

void
bt_endhostent(*void*);

*struct protoent **
bt_getprotobyname(*const char *name*);

*struct protoent **
bt_getprotobynumber(*int proto*);

*struct protoent **
bt_getprotoent(*void*);

void
bt_setprotoent(*int stayopen*);

void
bt_endprotoent(*void*);

int
bt_aton(*const char *str, bdaddr_t *ba*);

*const char **
bt_ntoa(*const bdaddr_t *ba, char *str*);

int
bt_devaddr(*const char *devname, bdaddr_t *addr*);

int
bt_devname(*char *devname, const bdaddr_t *addr*);

int
(bt_devenum_cb_t)(*int s, struct bt_devinfo const *di, void *arg*);

int
bt_devinfo(*struct bt_devinfo *di*);

int
bt_devenum(*bt_devenum_cb_t *cb, void *arg*);

int
bt_devopen(*char const *devname*);

int
bt_devclose(*int s*);

int
bt_devsend(*int s, uint16_t opcode, void *param, size_t plen*);

ssize_t
bt_devrecv(*int s, void *buf, size_t size, time_t to*);

int
bt_devreq(*int s, struct bt_devreq *r, time_t to*);

int

bt_devfilter(*int s, struct bt_devfilter const *new, struct bt_devfilter *old*);

void
bt_devfilter_pkt_set(*struct bt_devfilter *filter, uint8_t type*);

void
bt_devfilter_pkt_clt(*struct bt_devfilter *filter, uint8_t type*);

int
bt_devfilter_pkt_tst(*struct bt_devfilter const *filter, uint8_t type*);

void
bt_devfilter_evt_set(*struct bt_devfilter *filter, uint8_t event*);

void
bt_devfilter_evt_clt(*struct bt_devfilter *filter, uint8_t event*);

int
bt_devfilter_evt_tst(*struct bt_devfilter const *filter, uint8_t event*);

int
bt_devinquiry(*char const *devname, time_t length, int num_rsp, struct bt_devinquiry **ii*);

*char **
bt_devremote_name(*char const *devname, const bdaddr_t *remote, time_t to, uint16_t clk_off, uint8_t ps_rep_mode, uint8_t ps_mode*);

*char **
bt_devremote_name_gen(*char const *btooth_devname, const bdaddr_t *remote*);

int
bdaddr_same(*const bdaddr_t *a, const bdaddr_t *b*);

int
bdaddr_any(*const bdaddr_t *a*);

int
bdaddr_copy(*const bdaddr_t *dst, const bdaddr_t *src*);

DESCRIPTION

The **bt_gethostent**(), **bt_gethostbyname**() and **bt_gethostbyaddr**() functions each return a pointer to an object with the *hostent* structure describing a Bluetooth host referenced by name or by address, respectively.

The *name* argument passed to **bt_gethostbyname**() should point to a NUL-terminated hostname. The *addr* argument passed to **bt_gethostbyaddr**() should point to an address which is *len* bytes long, in binary form (i.e., not a Bluetooth BD_ADDR in human readable ASCII form). The *type* argument specifies the address family of this address and must be set to AF_BLUETOOTH.

The structure returned contains the information obtained from a line in */etc/bluetooth/hosts* file.

The **bt_sethostent**() function controls whether */etc/bluetooth/hosts* file should stay open after each call to **bt_gethostbyname**() or **bt_gethostbyaddr**(). If the *stayopen* flag is non-zero, the file will not be closed.

The **bt_endhostent**() function closes the */etc/bluetooth/hosts* file.

The **bt_getprotoent**(), **bt_getprotobyname**() and **bt_getprotobynumber**() functions each return a pointer to an object with the *protoent* structure describing a Bluetooth Protocol Service Multiplexor referenced by name or number, respectively.

The *name* argument passed to **bt_getprotobyname**() should point to a NUL-terminated Bluetooth Protocol Service Multiplexor name. The *proto* argument passed to **bt_getprotobynumber**() should have numeric value of the desired Bluetooth Protocol Service Multiplexor.

The structure returned contains the information obtained from a line in */etc/bluetooth/protocols* file.

The **bt_setprotoent**() function controls whether */etc/bluetooth/protocols* file should stay open after each call to **bt_getprotobyname**() or **bt_getprotobynumber**(). If the *stayopen* flag is non-zero, the file will not be closed.

The **bt_endprotoent**() function closes the */etc/bluetooth/protocols* file.

The **bt_aton**() routine interprets the specified character string as a Bluetooth address, placing the address into the structure provided. It returns 1 if the string was successfully interpreted, or 0 if the string is invalid.

The routine **bt_ntoa**() takes a Bluetooth address and places an ASCII string representing the address into the buffer provided. It is up to the caller to ensure that provided buffer has enough space. If no buffer was provided then internal static buffer will be used.

The **bt_devaddr**() function interprets the specified *devname* string as the address or device name of a Bluetooth device on the local system, and places the device address in the provided *bdaddr*, if any. The function returns 1 if the string was successfully interpreted, or 0 if the string did not match any local device. The **bt_devname**() function takes a Bluetooth device address and copies the local device name associated with that address into the buffer provided, if any. Caller must ensure that provided buffer is at least HCI_DEVNAME_SIZE characters in size. The function returns 1 when the device was found, otherwise 0.

The **bt_devinfo**() function populates provided *bt_devinfo* structure with the information about given Bluetooth device. The caller is expected to pass Bluetooth device name in the *devname* field of the passed *bt_devinfo* structure. The function returns 0 when successful, otherwise -1. The *bt_devinfo* structure is defined as follows

```
struct bt_devinfo
{
        char            devname[HCI_DEVNAME_SIZE];

        uint32_t        state;

        bdaddr_t        bdaddr;
        uint16_t        _reserved0;

        uint8_t         features[HCI_DEVFEATURES_SIZE];

        /* buffer info */
        uint16_t        _reserved1;
        uint16_t        cmd_free;
        uint16_t        sco_size;
        uint16_t        sco_pkts;
        uint16_t        sco_free;
        uint16_t        acl_size;
        uint16_t        acl_pkts;
        uint16_t        acl_free;

        /* stats */
        uint32_t        cmd_sent;
        uint32_t        evnt_recv;
        uint32_t        acl_recv;
        uint32_t        acl_sent;
        uint32_t        sco_recv;
```

```
        uint32_t        sco_sent;
        uint32_t        bytes_recv;
        uint32_t        bytes_sent;

        /* misc/specific */
        uint16_t        link_policy_info;
        uint16_t        packet_type_info;
        uint16_t        role_switch_info;
        uint16_t        debug;

        uint8_t         _padding[20];
};
```

The **bt_devenum**() function enumerates Bluetooth devices present in the system. For every device found, the function will call provided *cb* callback function which should be of *bt_devenum_cb_t* type. The callback function is passed a HCI socket *s*, fully populated *bt_devinfo* structure *di* and *arg* argument provided to the **bt_devenum**(). The callback function can stop enumeration by returning a value that is greater than zero. The function returns number of successfully enumerated devices, or -1 if an error occurred.

The **bt_devopen**() function opens a Bluetooth device with the given *devname* and returns a connected and bound HCI socket handle. The function returns -1 if an error has occurred.

The **bt_devclose**() closes the passed HCI socket handle *s*, previously obtained with bt_devopen(3).

The **bt_devsend**() function sends a Bluetooth HCI command with the given *opcode* to the provided socket *s*, previously obtained with bt_devopen(3). The *opcode* parameter is expected to be in the host byte order. The *param* and *plen* parameters specify command parameters. The **bt_devsend**() function does not modify the HCI filter on the provided socket *s*. The function returns 0 on success, or -1 if an error occurred.

The **bt_devrecv**() function receives one Bluetooth HCI packet from the socket *s*, previously obtained with bt_devopen(3). The packet is placed into the provided buffer *buf* of size *size*. The *to* parameter specifies receive timeout in seconds. Infinite timeout can be specified by passing negative value in the *to* parameter. The **bt_devrecv**() function does not modify the HCI filter on the provided socket *s*. The function returns total number of bytes received, or -1 if an error occurred.

The **bt_devreq**() function makes a Bluetooth HCI request to the socket *s*, previously obtained with bt_devopen(3). The function will send the specified command and will wait for the specified event, or timeout *to* seconds to occur. The *bt_devreq* structure is defined as follows

```
struct bt_devreq
{
        uint16_t        opcode;
        uint8_t         event;
        void            *cparam;
        size_t          clen;
        void            *rparam;
        size_t          rlen;
};
```

The *opcode* field specifies the command and is expected to be in the host byte order. The *cparam* and *clen* fields specify command parameters data and command parameters data size respectively. The *event* field specifies which Bluetooth HCI event ID the function should wait for, otherwise it should be set to zero. The HCI Command Complete and Command Status events are enabled by default. The *rparam* and *rlen* parameters specify buffer and buffer size respectively where return parameters should be placed. The **bt_devreq**() function temporarily modifies filter on the provided HCI socket *s*. The function returns 0 on success, or -1 if an error occurred.

The **bt_devfilter**() controls the local HCI filter associated with the socket *s*, previously obtained with bt_devopen(3). Filtering can be done on packet types, i.e. ACL, SCO or HCI, command and event packets, and, in addition, on HCI event IDs. Before applying the *new* filter (if provided) the function will try to obtain the current filter from the socket *s* and place it into the *old* parameter (if provided). The function returns 0 on success, or -1 if an error occurred.

The **bt_devfilter_pkt_set**(), **bt_devfilter_pkt_clr**() and **bt_devfilter_pkt_tst**() functions can be used to modify and test the HCI filter *filter*. The *type* parameter specifies HCI packet type.

The **bt_devfilter_evt_set**(), **bt_devfilter_evt_clr**() and **bt_devfilter_evt_tst**() functions can be used to modify and test the HCI event filter *filter*. The *event* parameter specifies HCI event ID.

The **bt_devinquiry**() function performs Bluetooth inquiry. The *devname* parameter specifies which local Bluetooth device should perform an inquiry. If not specified, i.e. NULL, then first available device will be used. The *length* parameters specifies the total length of an inquiry in seconds. If not specified, i.e. 0, default value will be used. The *num_rsp* parameter specifies the number of responses that can be received before the inquiry is halted. If not specified, i.e. 0, default value will be used. The *ii* parameter specifies where to place inquiry results. On success, the function will return total number of inquiry results, will allocate, using calloc(3), buffer to store all the inquiry results and will return pointer to the allocated buffer in the *ii* parameter. It is up to the caller of the function to dispose of the buffer using free(3) call. The function returns -1 if an error has occurred. The *bt_devinquiry* structure is defined as follows

```
struct bt_devinquiry {
        bdaddr_t        bdaddr;
        uint8_t         pscan_rep_mode;
        uint8_t         pscan_period_mode;
        uint8_t         dev_class[3];
        uint16_t        clock_offset;
        int8_t          rssi;
        uint8_t         data[240];
};
```

The **bt_devremote_name**() function performs Bluetooth Remote Name Request procedure to obtain the user-friendly name of another Bluetooth unit. The *devname* parameter specifies which local Bluetooth device should perform the request. If not specified (NULL), the first available device is used. The *remote* parameter specifies the Bluetooth BD_ADDR of the remote device to query. The *to* parameter specifies response timeout in seconds. If not specified (0), the default value is taken from the net.bluetooth.hci.command_timeout sysctl(8) value. The *clk_off*, *ps_rep_mode*, and *ps_mode* parameters specify Clock_Offset, Page_Scan_Repetition_Mode, and Page_Scan_Mode fields of HCI_Remote_Name_Request respectively. On success, the function returns a pointer to dynamically allocated NUL-terminated string or NULL if an error occurred. It is up to the caller to release returned string using free(3).

The **bt_devremote_name_gen**() function is a shortcut to **bt_devremote_name**() that passes generic defaults for *to*, *clk_off*, *ps_rep_mode*, and *ps_mode* parameters.

The **bdaddr_same**(), **bdaddr_any**(), and **bdaddr_copy**() are handy shorthand Bluetooth address utility functions. The **bdaddr_same**() function will test if two provided BD_ADDRs are the same. The **bdaddr_any**() function will test if provided BD_ADDR is ANY BD_ADDR. The **bdaddr_copy**() function will copy provided *src* BD_ADDR into provided *dst* BD_ADDR.

FILES

/etc/bluetooth/hosts
/etc/bluetooth/protocols

EXAMPLES

Print out the hostname associated with a specific BD_ADDR:

```
const char *bdstr = "00:01:02:03:04:05";
bdaddr_t bd;
struct hostent *hp;
```

```
if (!bt_aton(bdstr, &bd))
        errx(1, "can't parse BD_ADDR %s", bdstr);

if ((hp = bt_gethostbyaddr((const char *)&bd,
   sizeof(bd), AF_BLUETOOTH)) == NULL)
        errx(1, "no name associated with %s", bdstr);

printf("name associated with %s is %s\n", bdstr, hp->h_name);
```

DIAGNOSTICS

Error return status from **bt_gethostent**(), **bt_gethostbyname**() and **bt_gethostbyaddr**() is indicated by return of a NULL pointer. The external integer *h_errno* may then be checked to see whether this is a temporary failure or an invalid or unknown host. The routine herror(3) can be used to print an error message describing the failure. If its argument *string* is non-NULL, it is printed, followed by a colon and a space. The error message is printed with a trailing newline.

The variable *h_errno* can have the following values:

HOST_NOT_FOUND No such host is known.

NO_RECOVERY Some unexpected server failure was encountered. This is a non-recoverable error.

The **bt_getprotoent**(), **bt_getprotobyname**() and **bt_getprotobynumber**() return NULL on EOF or error.

SEE ALSO

gethostbyaddr(3), gethostbyname(3), getprotobyname(3), getprotobynumber(3), herror(3), inet_aton(3), inet_ntoa(3), ng_hci(4)

CAVEAT

The **bt_gethostent**() function reads the next line of */etc/bluetooth/hosts*, opening the file if necessary.

The **bt_sethostent**() function opens and/or rewinds the */etc/bluetooth/hosts* file.

The **bt_getprotoent**() function reads the next line of */etc/bluetooth/protocols*, opening the file if necessary.

The **bt_setprotoent**() function opens and/or rewinds the */etc/bluetooth/protocols* file.

The **bt_devenum**() function enumerates up to HCI_DEVMAX Bluetooth devices. During enumeration

the **bt_devenum**() function uses the same HCI socket. The function guarantees that the socket, passed to the callback function, will be bound and connected to the Bluetooth device being enumerated.

AUTHORS

Maksim Yevmenkin <*m_evmenkin@yahoo.com*>

BUGS

Some of those functions use static data storage; if the data is needed for future use, it should be copied before any subsequent calls overwrite it.